Project: Real-Time Trading Platform with Order Book Matching

Scenario Overview

Build a production-grade, real-time trading web application that lets users register, securely place trades, and view a live order book and trade history. The system must handle complex order matching, real-time updates, concurrent transactions, and robust security.

To-Do List / Candidate Tasks

1. Set Up Project Infrastructure

- [ ] Initialize a monorepo with backend FastAPI , frontend Streamlit or React), and database PostgreSQL/MySQL .
- [ ] Set up Docker Compose for running all services locally.
- [ ] Prepare a requirements.txt or pyproject.toml for dependencies.

2. User Management & Security

- [ ] Implement user registration and login endpoints.
- [ ] Secure authentication using JWT tokens.
- [ ] Hash passwords safely using bcrypt or passlib.
- [ ] Add role-based authorization (trader/admin).
- [ ] Enforce HTTPS (self-signed certificate for local development).
- [ ] Validate and sanitize all user inputs.

3. Order Book Engine & APIs

- [ ] Design core Python data structures for buy/sell order matching (market/limit orders).
- [ ] Implement back-end Python logic for real-time order matching and trade execution.
- [ ] Expose RESTful APIs:
    - [ ] Place new order (buy/sell).
    - [ ] Cancel order.

    - [ ] List active orders and user trade history.
- [ ] Write API documentation with OpenAPI/Swagger.

4. Real-Time Data Streaming

- [ ] Create a FastAPI WebSocket endpoint for live order book and trade feed.
- [ ] Test by subscribing with a simple Python WebSocket client.

5. Frontend Dashboard

- [ ] Build a web dashboard Streamlit, Dash, or React):
    - [ ] Show live order book, current trades, and user's personal trade history.
    - [ ] Include forms to place/cancel orders and to view notifications/alerts.
    - [ ] Authenticate users and restrict dashboard views by roles.

6. Persistence & Database Integration

- [ ] Set up PostgreSQL/MySQL with SQLAlchemy ORM.
- [ ] Store all users, orders, and trade history securely.
- [ ] Implement DB migration scripts.

7. Testing & Code Quality

- [ ] Write unit and integration tests for core logic and APIs (pytest).
- [ ] Achieve high test coverage (report in README .
- [ ] Run static analysis/formatting (black, flake8 .

8. DevOps & Deployment

- [ ] Provide a working Docker Compose config for all services.
- [ ] Document setup, architecture diagram, and troubleshooting in README.
- [ ] Optional) Add GitHub Actions for CI/CD: linting, tests, deployment.

9. Bonus Enhancements

- [ ] Add API rate limiting.
- [ ] Implement front-end notifications for completed/cancelled trades.
- [ ] Propose approaches for scaling backend concurrency (summary).

Submission Checklist

- [ ] Structured and well-documented source code.
- [ ] API docs and architecture diagram (in markdown or png).
- [ ] README with tech choices, setup steps, and test coverage.
- [ ] Docker Compose file for local deployment.
- [ ] Sample demo (screenshots or video walkthrough, optional).